2015

Model of distributed software development using system dynamics

Sourajit Ghosh Dastidar
Iowa State University

Model of distributed software development using system dynamics

by

Sourajit Ghosh Dastidar

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
David Weiss, Major Professor
Wallapak Tavanapong
Shashi Gadia

Iowa State University

Ames, Iowa

2015

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

My utmost gratitude go to those who have helped me with various aspects of my research and writing my thesis. Firstly, I thank my adviser Dr. David Weiss for his guidance, patience and support throughout my time at Iowa State University, conducting research and writing my thesis. I thank my research partner Ya Chen who has helped with data collection and provided valuable insights. I also thank my committee members for their efforts and contribution to this work: Dr. Shashi Gadia and Dr. Wallapak Tavanapong. I thank my friends Rui Ding and Manisha Rayanchu for helping and supporting me in all stages of my graduate career.

Thanks to the departmental staff of Computer Science at Iowa State University for being so helpful, approachable and supportive.

Finally, I am eternally grateful to my brother Shankhajit Ghosh, my parents Indrajit Ghosh-Dastidar and Namita Ghosh-Dastidar for their blessings, help, patience, support and advice throughout my life. I also know that my late grandmother Swarnomoyee Choudhury is watching over me and is always with me.

ABSTRACT

Distributed Software Development today is one of the most widely used and implemented software development strategies in the industry [1]. Some of the major advantages of this methodology are 24 hour work-cycle [2], increased diversity of resources, reduced labor costs, decreased time of iteration cycle and diverse skillset of the workforce [3]. Although it has proven to be quite efficient and practical, there are ample reasons from previous research [4][5] in this field that show that this development approach is uncertain in terms of quality of product developed, speed and expenses. Factors such as presence of multiple stakeholders, lack of effective communication among sites, cultural differences among the workforce and presence of a diverse range of system variables brings a level of uncertainty into the system. A method is required to simulate iterations of the software development lifecycle and understand the effect of changes in system variables/stakeholders involved. This would help project managers, business analysts and other parties involved from different sites to examine the effect of changes in one variable at any point to the other variables and inspect its short and long term consequence on the project plan and deliverables. Problems leading to faulty product development, failure in conforming to all the lifecycle requirements, decreased customer satisfaction, unforeseen expenses and inability to meet deadlines can be avoided by predicting changes using those predictions to make better decisions.

In this thesis, I have created a simulated model of Distributed Software Development using the concept of System Dynamics [6]. My main purpose is to

define the different variables, and stakeholders involved in this methodology. Furthermore, I aim to define relationships among them, analyze and draw sufficient conclusions that would help understand and decrease uncertainty. As an example, the results of the simulation show prediction of change of the number of customers, features released with time for the given product as other variables in the system change. This can help project directors, managers and leads to make better informed decisions about the steps they can take to maximize their product growth in the market.

CHAPTER 1. INTRODUCTION

1.1 Distributed Software Development

Distributed Software Development (DSD) is a development process and associated environment where the development is structured in such a way that it allows individual components/modules to be designed and implemented at different sites. These sites are often situated across different countries separated by multiple time zones and cultures. The main difference between DSD and traditional methods of software development is that different sites have to collaborate across large distances. Thus it takes a considerable effort to make sure that everyone is apprised of what the interacting sites are doing and that they are synchronizing. Extra effort is taken in making sure that the problems occurring from lack of proper communication resulting from cultural and linguistic barriers are resolved early. Since the sites are separated by various boundaries such as contextual, organizational, temporal, geographical and political, new unknown problems arise in managing such a project. Despite these problems, DSD has emerged as the most widely used model of development in the modern world [1]**.** Reasons include ability to work beyond regular hours [2], lower software development costs at offshore centers situated in emerging economies, the diverse capability and skillset of human resources in other regions and ability to reach markets well beyond national boundaries. Some countries in parts of the world might have regulations that only allow products

developed using in-house facilities to be marketed. Distributed development is particularly useful in such cases. Previous papers and research work have addressed the need for a standardized practice methodology, proper modularization of system architecture, presence of consistent/effective ways to define module interface specification, regular communication among sites, frameworks to decrease errors occurring because of lack of communication, multiple communication media and presence of liaisons [7]. What remains to be confronted is the lack of ways of predicting effects of changes in system variables and stakeholders where development is distributed among multiple sites. This brings uncertainty into the system that might hinder progress in project plans, leading to poor quality of the features developed, ultimately leading to more defects in the developed product than originally predicted and inability to meet design requirements/specifications resulting in loss of customers and other resources. This problem can be addressed using System Dynamics [6] which can be used to create simulated models. The main purpose of my research is to create a simulation involving relevant system variables and stakeholders, define relationships among them. This would help anticipate impacts, thereby allowing stakeholders/participants to take actions to deal with these problems without changing the originally envisioned deliverables and their deadlines.

1.2 Organization

The rest of the thesis is organized as follows: Chapter 2 starts with a brief introduction of the system dynamics modelling approach and its applications. In

Chapter 3, I start by introducing the case study. Chapter 4 talks about the data collection method and technique we applied to inspect the case study and obtain information. Chapter 5 introduces the system dynamics model of distributed development by talking about the System Variables, Causal Loop Diagram, Stock Flow Diagram and Equations. Chapter 6 presents the results, analyzes the model that we have created and talk about the challenges faced. Finally in Chapter 7, I conclude my research work and discuss ideas about how the model can be improved further in the future.

CHAPTER 2. SYSTEM DYNAMICS BASICS

System Dynamics has been extensively used to model complex feedback systems in various domains including population, ecological, economic and education systems [9] [15] [16] [17]. It has only been used a few times to model software development methodologies or systems [8] [14].

In System Dynamics, we start by defining a system that is a collection of variables or subsystems that interact in such a way that the whole system has properties that are not evident from analyzing the parts themselves [8]. When an event occurs that influences behavior of a variable or a set of variables, it also impacts other variables that are related to it along with itself in the future. Identifying and analyzing such a situation is done using feedback loops or causal loops. System Dynamics also looks at the whole system as an interaction of various subparts and assumes that the behavior cannot be explained in only terms of dynamics of the sub parts. It uses computer simulation to understand, analyze complex systems and effects of impromptu events on the future of the whole system. Using System Dynamics, emergent properties such as productivity and performance of product in the market that can only be evaluated or calculated after the structure of the whole system has been identified, and the interactions among its parts are known.

2.1 Causal Links

A causal loop diagram is a group of nodes representing the variables that are connected through cause and effect relationships. The relationships among variables are indicated by arrows (called causal links). An arrow from variable A to variable B means that any change in A would lead to changes in B either instantaneously or after a certain period of time. Depending on the relationship between the cause and effect, arrows are labeled as either positive or negative.

- A positive causal link (represented by a + sign on the link) indicates that if a variable increases, the variable on the positive enforcement side also increases and if the variable decreases then the other also decreases. In the example given in figure 2.1, if the quality of modularization during the design and architecture phase of a product development affects the quality of the module interface specifications. Better modularization leads to better interface specifications.

Figure 2.1 Positive Causal Link

- A negative causal link (represented by a – sign on the link) indicates that if a variable increases, the variable on the negative enforcement side

decreases and if the variable decreases then the other increases. In figure 2.2, increasing schedule pressure during an iteration leads to stricter deadlines and thus decreases the percentage of requirements that can be fulfilled.

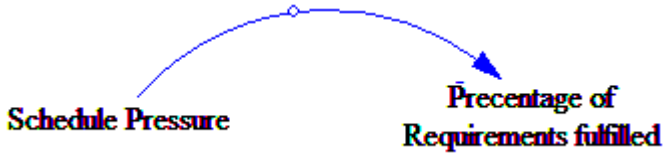

Figure 2.2 Negative Causal Link

2.2 Stock Flow Diagram (SFD)

A stock flow diagram shows dependencies among different variables that have a potential to change over time. An SFD consist of four types of elements: stock variables, auxiliary variables, flows and information.

Stock variable is an entity (boxed variable) whose value increases over time from inflows and decreases from outflows. Stocks are changed only by flows into the system and out of the system. Stocks normally have a certain value at each moment of time. In figure 2.3, the number of customers using a certain product at any given time is represented by the stock variable **Customers**.

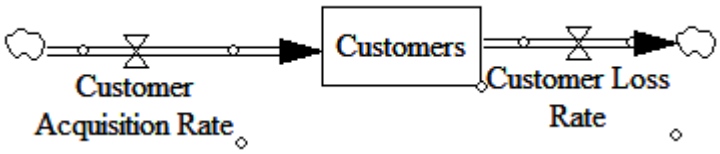

Figure 2.3 Customer Stock Variable

An **auxiliary variable** is represented by a point or named constant in the system. It is the subsystems/resources/deliverables present in the environment that assists in accumulation or depletion of stock as well as other auxiliary variables. In figure 2.4, the Number of Distributed Sites is an auxiliary variable whose increase leads to an increase in the communication problem rate thus increasing the **Communication Overhead** stock variable. Cultural Similarity among sites is also an auxiliary variable whose increase leads to an increase in Resolution Rate of Communication Problems and thus decreases the stock variable Communication Overhead.

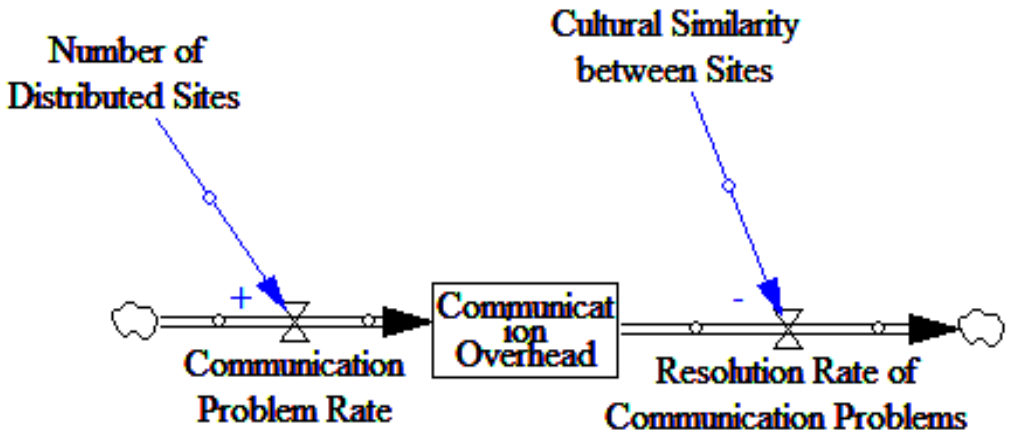

Figure 2.4 Communication Overhead stock variable along with different Auxiliary variables

A Flow changes a stock variable over time. Inflows add to the stock and outflows subtract from or deplete the stock variable. Flows are typically measured over a certain period of time. In figure 2.4, Communication Problem Rate is an inflow into Communication Overhead stock variable. Communication Problem Rate determines the number of problems occurring because of a lapse in communication among teams over a week or a month. Resolution Rate of

Communication Problems is a number representing resolution rate of problems caused by miscommunication over a unit of time.

Finally, **Link** between Extent of Requirements being Fulfilled and Schedule pressure is shown by a curved arrow in figure 2.5 and means that in some way, information about the value of one variable changes the value of another variable.

Figure 2.5 Link between two variables

2.3 Equations

Stock flow diagrams are used to represent graphically relationships of a system, and its variables to give proper insights about the different processes and interactions in the system. We need to determine how subsystems react with each other with respect to their dependencies. In order to analyze and determine the behavior of a system, we quantify each variable and also create relationships among them. These relationships are defined using equations. A Stock Flow Diagram enables formulation of these links among variables using differential and integral equations over a certain period of time. It quickly becomes impractical to solve such equations by hand as the values of the variables increases and the relationships among them become more complicated over time. For example, the number of communication problems at any instant of time t is equal to the initial number of Communication Problems plus the number of Communication Problems that were added minus the number of Communication problems that were solved. If inflow and outflow is measured in problems per unit time and their values are Inflow and Outflow respectively.

Number of Communication Overhead Problems =

$$Initial\ Communication\ Overhead\ Problems\ +\ \int_0^T (Inflow - Outflow)dt$$

2.4 Simulation

We start by creating a stock and flow diagram for the DSD environment using one of the many commercial software packages available such as VenSim, Stella, PowerSim etc. We then insert initial values that we have procured from past and current distributed software projects for various stocks into the model and also for the equations of the flows/links. Thereafter, the software package solves the relationships determined by the equations. The result is a time history for each of the variables in the model provided that there are no errors in the equations or relationships identified. The time history can either be shown in a graphical or tabular form. Users can run their "what if" policies to test real world scenarios thus providing them with clearer understanding of changes over time. This helps analyze the behavior of the system and determine alternative policies and their impacts [9]

For our model we have used the VenSim PLE simulation package [10] since it is easy to use and also has all the tools required for simulating a distributed environment.

CHAPTER 3. RELATED WORK

3.1 Introduction to Case Study

In this chapter, we introduce the case study and the data set that was used in order to understand distributed software development methodology. We also discuss the different components, stakeholders, advantages and potential problems involved. Iowa State University currently has a course project conducted over a period of four months, offered each year in the fall semester. The project is a collaborative effort among students and faculties from Iowa State University (ISU-Ames, United States), Ji Lin University (JLU-Chang Chun, China), King Mongkut's University of Technology (KMUTT-Bangkok, Thailand) and National University of Columbia (UNAL-Bogota, Columbia) (Participating universities changed from year to year but were the same in the period of this study). The course was offered independently by each university. The students developed a software application by working collaboratively over different geographic locations, time zones, cultures, and languages. Various teams were observed and data were collected over a two year interval (the Fall 2013, Fall 2014 instances of the course) under careful inspection.

The main purpose of the project was to create a Class Room Face Recognition System (CFRS) for instructors in a class who would like help in guiding class interactions, particularly in issues such as remembering student names, associating names with faces and keeping track of attendance. As shown in figure 3.1 CFRS uses a camera and computer/mobile device to identify students in the

classroom and to check and record attendance automatically. A typical client application takes a picture using a camera installed in the classroom and sends the picture for identification to a remote server. The server refers the class picture to its student database and uses the face recognition algorithm from Open CV.org/Face.com or others to identify the faces in the picture and respond back to the client with student names and other information. The client then updates the student attendance table and displays the students present for that session. The client application could be used by both students and teachers. The teachers use their client application to have a summarized attendance of the class, while students can use it to review their own attendance for classes. Teachers can also set alarms to remind students if their attendance is low in a particular course.

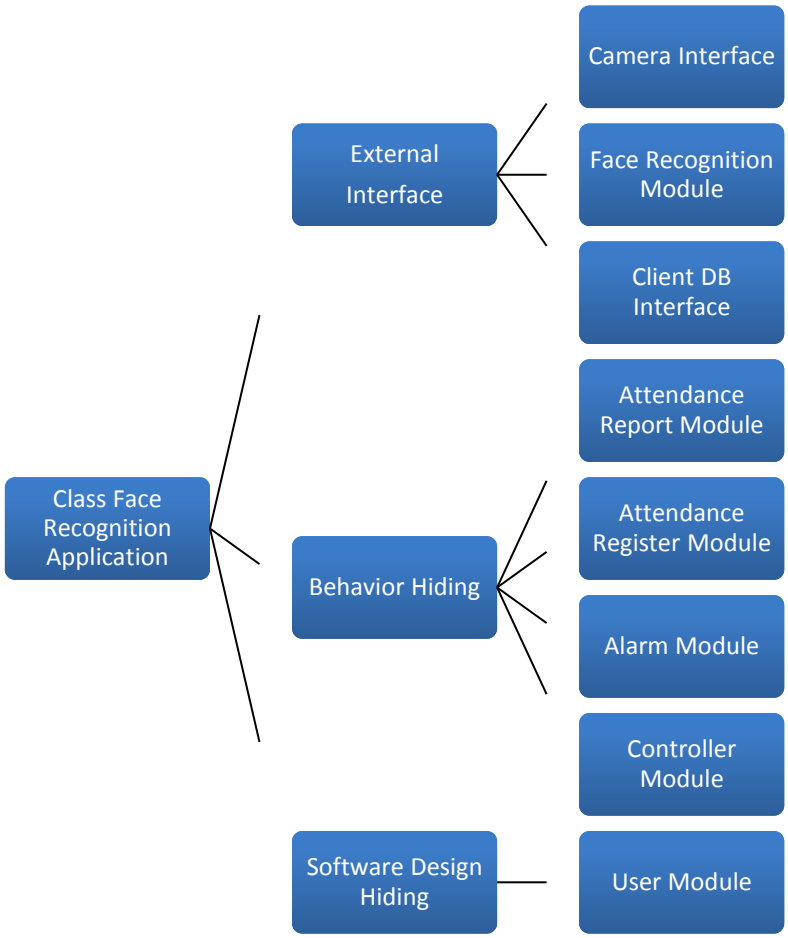

Figure 3.1 Context diagram showing Face Recognition System Module Architecture (module structure is adapted from [21])

3.2 Deliverables and Tools

Throughout the whole duration of the project each team was independently required to produce a strict set of deliverables as in a real world software project. Various members in a team had ownership of each deliverable and had to produce them following stringent deadlines. This required proper synchronization of work among the different teams. Tables 3.1, 3.2 give the list of deliverables that the teams at Iowa State University were required to produce in the Fall 2014 and Fall 2013 iterations of the project respectively.

Table 3.1 Deliverable, Deadlines and Roles Responsible for project iteration in Fall 2014

Artifacts	Date Due	Role Responsible for Artifact
Team name and role designations (Team Composition Form)	2 Sep 2014	Project Manager and Liaison
Requirements: Initial Context diagram	4 Sep 2014	Systems Engineer
Initial Project Plan, including risks, wbs, project measures	9 Sep 2014	Systems Engineer
First Prototype	11 Sep 2014	Systems Engineer
Requirements: Initial Use Cases and Test Cases	11 Sept 2014	Systems Engineer
Configuration management policy (coordinate policy with distributed teams, ensure build works)	16 Sept 2014	Systems Engineer
Requirements: Initial Output Specifications	18 Sept 2014	Systems Engineer
Initial Module Structure	22 Sep 2014	Architect
Initial Uses Structure	22 Sep 2014	Architect
Initial Process Structure(s),	25 Sep 2014	Architect
Second Prototype*	2 Oct 2014	Systems Engineer
First Set of Interface Specifications, including black-box test cases for each module	7 Oct 2014	Architect
Data, e.g. pictures and profile information, to be used for first system test cases	7 Oct 2014	Systems Engineer
Revised Module, Uses, Process Structures	9 Oct 2014	Architect
Revised project plan, with initial values for,	9 Oct 2014	Project
and analysis of, measures		Manager and
Second Set of Interface Specifications,	16 Oct 2014	Architect
First Module Implementations (corresponding	16 Oct 2014	Developer ***
System verification plan (using first data from	16 Oct 2014	Tester &
First module and integration test results	23 Oct 2014	Tester &
Second Module Implementations	28 Oct 2014	Developer
Data to be used for final system test cases	30 Oct 2014	Systems
Revised Implementations	6 Nov 2014	Developer
Second module and integration test results	11 Nov 2014	Tester &
System test results report	2 Dec 2014	Tester &
All-sites meeting for integration testing.	5 Dec 2014	
Retrospective Report	11 Dec 2014	Project
Final project presentation	11 Dec 2014	All

Table 3.2 Deliverable, Deadlines and Roles Responsible for project iteration in Fall 2013

Role Responsible for Artifact	Artifacts for which the role is responsible	Date Due
Systems Engineer	Requirements: Initial Context diagram	5 Sep 2013
	First Prototype	12 Sep 2013
	Requirements: Initial Use Cases	12 Sept 2013
	Requirements: Initial Output Specifications	19 Sept 2013
	Configuration management policy (coordinate policy with distributed teams, ensure build works)	17 Sept 2013
	Second Prototype*	3 Oct 2013
	Data, e.g. pictures and profile information, to be used for first system test cases	8 Oct 2013
	Data to be used for final system test cases	31 Oct 2013
Architect	Initial Module Structure	24 Sep 2013
	Initial Uses Structure	24 Sep 2013
	Initial Process Structure(s),	26 Sep 2013
	First Set of Interface Specifications	3 Oct 2013
	Revised Module, Uses, Process Structures	8 Oct 2013
	Second Set of Interface Specifications, including revised first set	10 Oct 2013
Developer	First Module Implementations (corresponding to first interface specifications)	17 Oct 2013
	Second Module Implementations (corresponding to second interface specifications)	29 Oct 2013
	Revised Implementations	7 Nov 2013

Table 3.2 (continued)

Tester & Integrator	System generation and verification plan (using first data from systems engineer)	17 Oct 2013
	First module and integration test results	24 Oct 2013
	Second module and integration test results	12 Nov 2013
	System test results report	19 Nov 2013
Project Manager and Liaison	Team name and role designations (Team Composition Form)	3 Sep 2013
	Initial Project Plan, including risks, wbs, project measures	10 Sep 2013
	Revised project plan, with initial values for, and analysis of, measures	10 Oct 2013
	Retrospective Report	12 Dec 2013
	Final project presentation	12 Dec 2013

All the hardware and software tools that would be required in the development lifecycle were outlined at the start of the project. Two Panasonic BB-HCM and BLC series cameras were used for taking pictures and providing live video feed of the classroom during testing. XCode and Netbeans IDE were used while implementing the modules. Git was used for source code management among various teams along with Assembla which provided repository hosting, file management, source code management and bug trafficking. Tools and applications such as Assembla Forums, Emails, Internet Chat and Video Calls were used by teams to synchronize work with each other, seek clarifications and provide support.

CHAPTER 4. DATA COLLECTION

4.1 Method

Data collection was an important aspect during the whole project. It enabled us to understand and get more insights about the distributed nature of the project. Although, the participants of the project were students who have lesser software development experience compared with many industrial developers, the data collection allowed us to understand how this course project emulated real world development. The data to be collected were identified using the Goal Question Metric approach [11] [12] [13]. This method assumes that in order to collect relevant data about the project, an organization must first define goals for the project, then trace that to the data that can be used to assess progress towards achieving those goals. As defined [11], the measurement model has three levels:

- Conceptual Level or Goal: "A goal is defined at the start of the project with reference to objects of measurements that can be Products such as artifacts, deliverables and documents, Processes such as specifying, designing and testing and Resources such as personnel, hardware, software etc."
- Operational Level or Question: Operational level is defined as "a set of questions that are defined to determine achievement of a specific goal."
- Quantitative Level or Metric: Quantitative level is defined as "the set of data or metrics that can be objective such as number of versions, number

of errors, lines of code and subjective such as readability of text, level of user satisfaction associated with every question in order to answer them quantitatively."

Some examples of the GQM model are given in figures 4.1 and 4.2.

Goal:	To measure effectiveness of communication among teams
Question:	How comprehensive were the meetings?
Metric:	Number of issues that were covered, Number of resulting action items

Figure 4.1 Goal Question Metric example

Goal:	To measure the cooperation and communication among teams
Question:	How good was the cooperation among teams in each phase of development?
Metric:	Number of issues that resulted from miscommunication among teams, Number of messages exchanged per development phase to solve existing issues (Requirement Analysis, Design, Implementation, Testing, Integration)

Figure 4.2 Goal Question Metric example continued

4.2 Source of Data

The data originated from the Distributed Software Development coursework over a period of 2 iterations of the course (Fall 2013, Fall 2014). Data were collected in the Fall 2013 and Fall 2014 course iterations under careful inspection. Figure 4.3 shows the composition of the teams in all the participating universities.

	2014 Fall		2013 Fall	
	Students	Teams	Students	Teams
ISU	18	3	10	2
JLU	12	2	3	1
UNAL	8	2	6	1
KMUTT	12	2	13	2

Figure 4.3 Composition of students at different universities

4.3 Validation

In order to ensure the accuracy of the data that was collected, participants were monitored weekly and under close inspection of instructors/professors at various sites. Students were required to complete a data collection form each week under strict guidelines. The data collection was a required part of the course. The major purpose was to educate students in how to collect software engineering data by teaching GQM method for measurement. Figure 4.4a, 4.4b shows the template of the data collection form that was used for this purpose.

Students were monitored and data that was collected was checked for consistency. Aberrations in data that were collected were investigated by conducting one on one interviews that helped understand and validate the reasons behind them. This also helped us understand how the project mirrored a real world environment and helped us to analyze the veracity of the information received.

Date	Week No.	Total Effort Expended (hours)	Interface Effort (hours)			
			Client/Facilitator	Facilitator/Server	Client/Database	Other

Figure 4.4a Data Collection Sheet Template Header

Communication Effort by Channel							Miscommunication Issues	
Face to Face (csite/local)	Cross-site chat sessions Number/effort (hours)	Local chat sessions Number/effort (hours)	Cross-site email Number/effort (hours)	Local email Number/effort (hours)	Assembla (csite/local)	Other (csite/local)	Issue ID	Effort to resolve (hours)

Figure 4.4b Data Collection Sheet Template Body

CHAPTER 5. SYSTEM DYNAMICS MODEL OF DISTRIBUTED SOFTWARE DEVELOPMENT

5.1 Introduction to System Variables

To model Distributed Software Development, we followed the iterative approach. First, we identified key variables that determine and change behavior in the system and their dependencies through which they affect each other. As an example, Quality of Modularization and Quality of Interface Specifications are two such variables. Improving the quality of modularization would result in improvement in interface specification because more comprehensible specifications can be written for modules that have well and clearly defined work assignments that do not change over time.

Furthermore, based on our understanding of the system, we identified important feedback loops and relations in the system among different variables as shown in the Causal Flow Diagram and then we made the equivalent Stock Flow diagram. Our list of variables for the system is given in Table 5.1. The stock variables of the system have been indicated in bold in the table.

Table 5.1 Data Dictionary of System Variables

Name	Description	Dimension
Available Trained Resources (Stock Variable)	The total number of trained resources/employees available for the project.	Number of People

Table 5.1 Data Dictionary of System Variables continued

Benched Recruits	Number of people recruited into the company/project but have not been trained to start directly contributing to the project	Number of People
Resource Training Rate	Rate at which resources from the Benched Recruits are given training to be moved onto Available Trained Resources	Number of People per Unit Time
Resource Relocation Rate	Rate at which resources from the Available Trained Resources pool are relocated from the project because of budget cuts or demands elsewhere or fired due to lack of productivity.	Number of People per Unit Time
Number of Distributed Sites	Number of Distributed Sites participating in the whole project. Sites can be in the same country separated by time zones and also in different Geographic locations having completely different work culture.	Number of Sites
Communication Overhead (Stock Variable)	Total open problems in the project that have arose because of gaps in Communication among different sites.	Number of Communication Problems

Table 5.1 Data Dictionary of System Variables continued

Number of Communication Media	Number of different communication media used by the sites to communicate with each other. (Ex: Teleconferencing, Video Calling, Email, Forum, Online chat)	Number of Media
Cultural Similarity among Sites	Similarity among the people working in different sites in terms of language, profession work ethic, Hofstede's Cultural Dimensions	Dimensionless (Qualitative variable measured on a scale of 0-1. 1 states that the cultures are perfectly at sync with one another)
Communication Problem Rate	Rate at which Communication problems arise per month	Communication Problems per Unit Time
Resolution Rate of Communication Problems	Rate at which communication problems are resolved per month	Number of Communication Problems per Unit Time
Customers (Stock Variable)	Customers of the product when it is available in the market	Number of People
Customer Acquisition Rate	Rate at which new customers are starting to use the product after it's release in the market	Number of People per Unit Time
Customer Loss Rate	Rate at which customers stop using the product after using it for sometime	Number of People per Unit Time
Number of Bugs	Number of bugs in the code per module before Quality Assurance and Testing	Number of Bugs

Table 5.1 Data Dictionary of System Variables continued

Number of Escaped Bugs	Number of bugs in the code after Quality Assurance and Testing. These bugs are in the released products that are being used by the customers.	Number of Bugs/Module
Rework	Extra effort required to be put in by developers to fix errors found during Quality Assurance and Testing.	Number of hours
Productivity	The efficiency of the team as a whole. It is measured as the number of logical changes made to the code base or the number of commits to the repository.	Number of Commits/Unit Time
Quality of Modularization	The quality of design of the overall system including organizing the system into individual work assignments. If the quality of modularization is good then no or few changes are made to the system and module structure over time.	Dimensionless (Qualitative variable measured on a scale of 0-1. 1 indicates proper modularization)
Quality of Interface Specification	The quality of interface specification of the application programming interfaces of the different modules. It is used to indicate that no/few errors occur over time due to	Dimensionless (Qualitative variable measured on a scale of 0-1. 1 indicates that the quality of interface specification is highest)

	interfacing and integration among modules.	

Table 5.1 Data Dictionary of System Variables continued

Features Under Development (Stock Variable)	The total number of features which have been put into the development phase to be added to the already existing product/new product.	Number of Features
New Feature Requests	New features that have been requested to be included In the new product being developed or already existing product by domain experts, customers and market demand.	Number of Features
Extent of Requirements Fulfilled	Percentage of original requirements (proposed at the start of the current development cycle) being fulfilled.	Percent/Fraction
Schedule Pressure	(Actual completion Time/Proposed Completion Time) Proposed Completion time is the number of days in which the current iteration was expected to be completed at the start of the iteration.	Dimensionless
Feature Released	The total number of features in the product that have been released in the market.	Number of Features.
Features pushed into Development Rate	Features that are pushed into	Number of Features Per Unit

	development per month. These features have been requested by the customer, domain experts or market demand.	Time

Table 5.1 Data Dictionary of System Variables continued

Feature Deletion Rate	Features that were originally pushed into development but have been taken off development due to schedule pressure, prevent escalating costs or low market demand.	Number of Features per Unit Time
Development Completion Rate	The number of completed features after development that are ready to be released in the market.	Number of Features per Unit Time

5.2 Causal Loop Diagram (CLD) of Distributed Software Development

Using the above mentioned variables and their relations with one another, we prepared a Causal Loop Diagram (CLD) in 3 steps. We have described the CLD in multiple steps in order to simplify the model.

- **Step 1 - Causal Loop Diagram with Features under Development and Number of Customers Stock Variables (Fig 5.1, 5.2):** We have defined the relationship among stock variables Customers and Features under Development along with their dependencies on other auxiliary variables as the first step. We have also identified the feedback loops and their polarities. For example, in figure 5.1 increase in the number of Customers

leads to increase in the Number of Bugs (before Quality Assurance and Testing). Note that the double line mark in the link between Customers and Number of Bugs denotes a delay in the effect. Increase in number of bugs leads to increase in the Number of Escaped Bugs (bug/errors in the system after Quality Assurance and Testing). Since the Number of Escaped Bugs are caught by customers using the product, it leads to lower customer satisfaction and decreases the Customer variable. The overall loop involving Customer -> Number of Bugs -> Number of Escaped Bugs -> Customers is a negative feedback loop since there are an odd number of negative causal links. The overall Casual Loop Diagram involving the Features under Development and Number of Customers stock variable is given in figure 5.2.

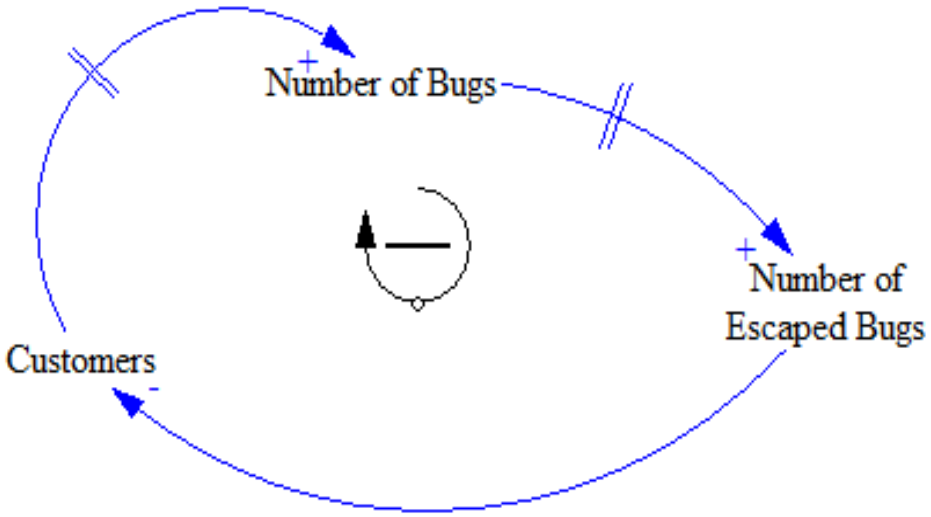

Figure 5.1 Negative Feedback Loop Example

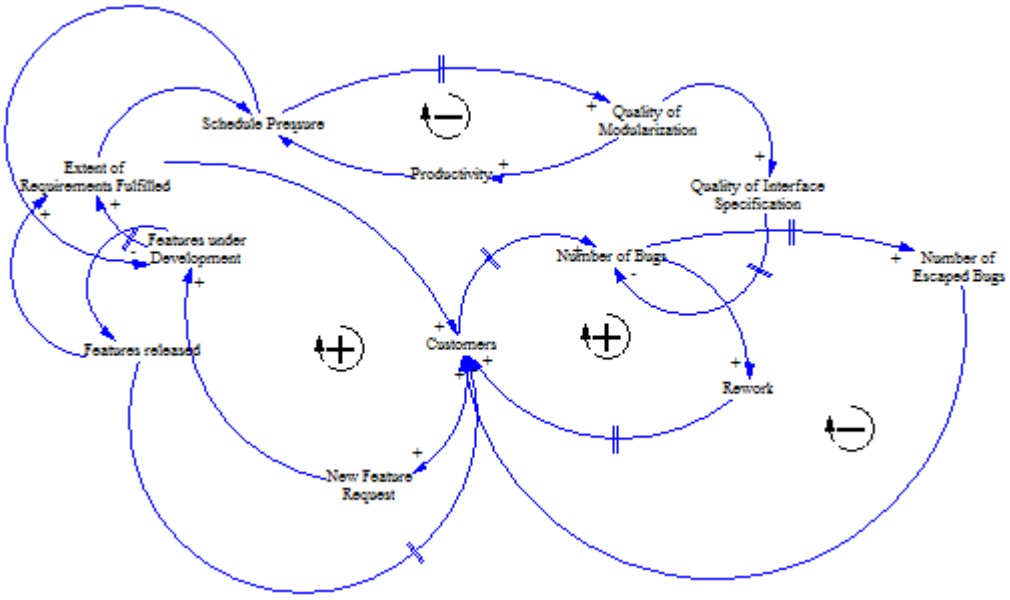

Figure 5.2 CLD involving Stock Variables Features under Development & Customers.

- **Step 2 – Causal Loop Diagram with Available Trained Resources and Communication Overhead Stock Variables (Fig 5.3):** In the second step, we define variables Available Trained Resources and Communication Overhead along with other auxiliary variables related to them. Note here that variables shown in Fig 5.3 have links/relationships with variables in Step 1 (Fig 5.1, Fig 5.2) that will be shown later. For example, Number of Benched Recruits increases the Number of Available Trained Resources after their training. Increase in Number of Distributed Sites leads to increase in Communication Overhead. Contrary to that, increase in Number of Communication Media leads to decrease in Communication Overhead and finally Increase in Cultural Similarity among Sites also leads to decrease in Communication Overhead.

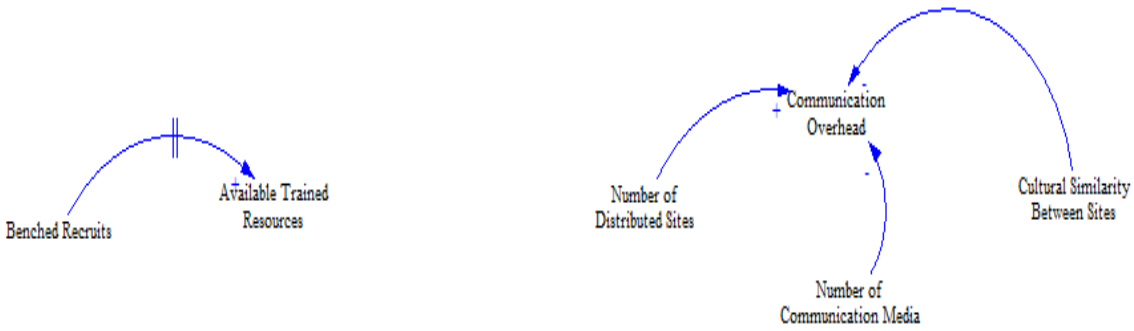

Figure 5.3 CLD involving stock variables Available Trained Resources and Communication Overhead.

- **Step 3 - Overall Causal Loop Diagram involving all variables (Fig 5.4):**

  In the final step we are defining relationships among all the stock variables and determining any further links among other auxiliary variables.

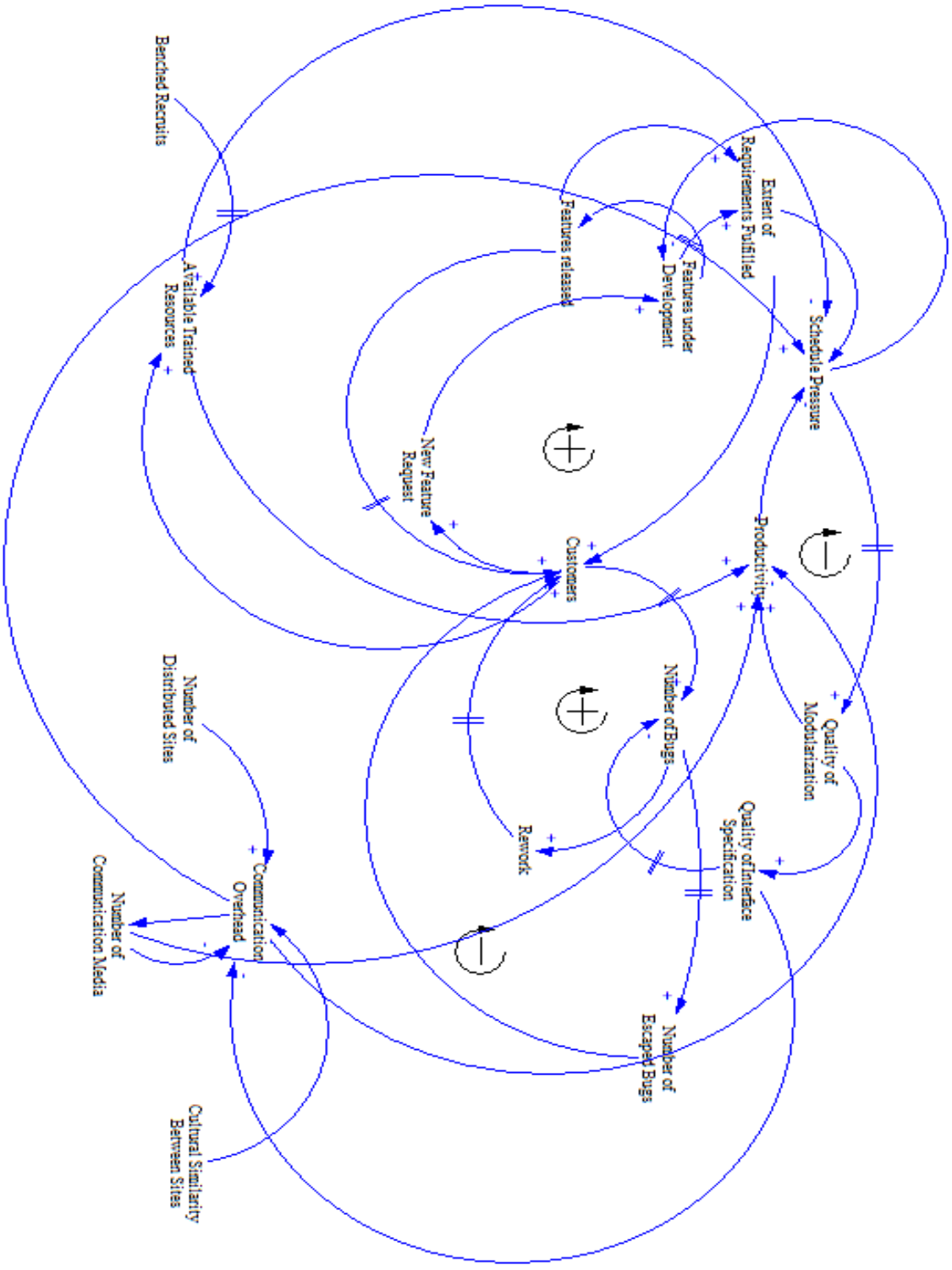

Figure 5.4 CLD involving all the environment variables.

5.3 Stock Flow Diagram of Distributed Software Development

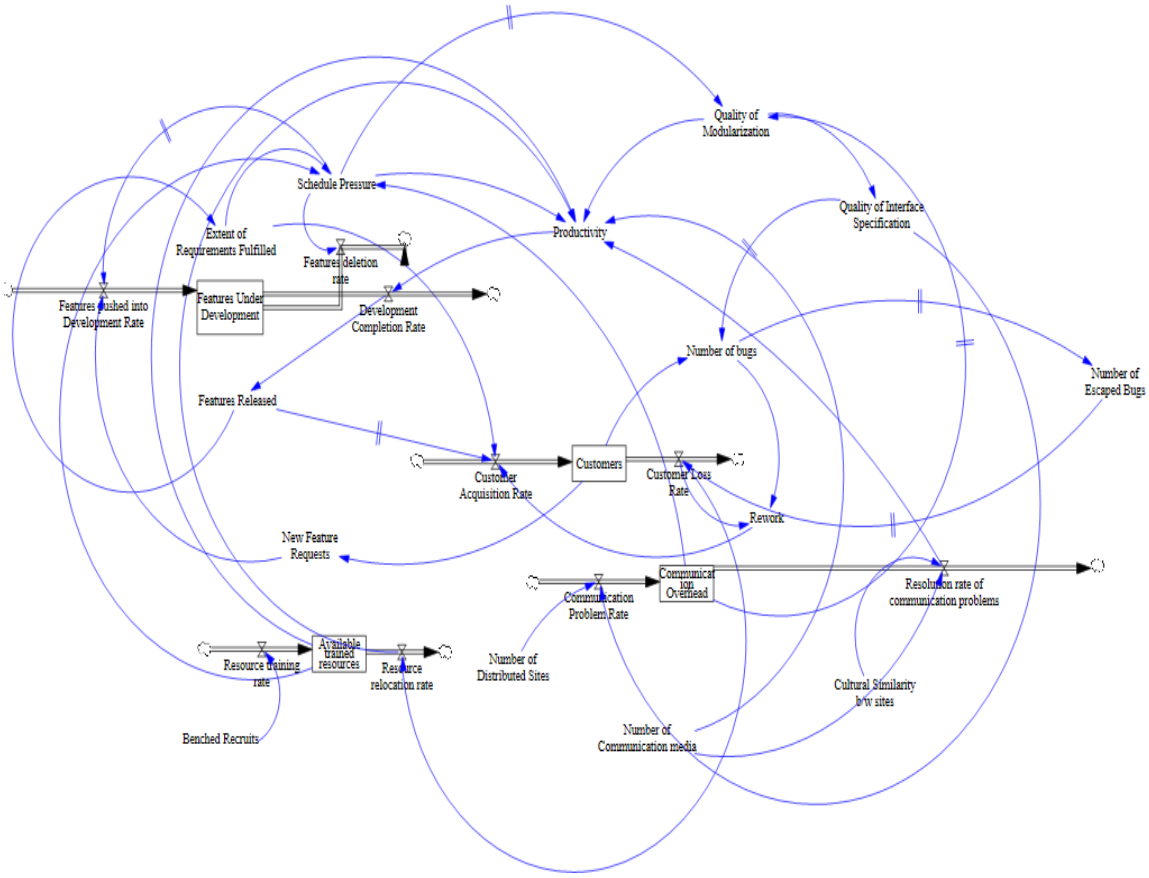

Figure 5.5 Stock Flow Diagram of Distributed Software Development

After preparing a Causal Loop Diagram (Fig 5.1, 5.2, 5.3, 5.4) of the environment, we produced a Stock Flow Diagram (Fig 5.5) involving the stock variables **Features Under Development**, **Customers**, **Available Trained Resources** and **Communication Overhead**. In order to simplify the stock flow diagram, some of the links have not been shown. The values of the stock variables are influenced by inflows and outflows. Each auxiliary variable that influences the inflow also influences the outflow in an opposite way. As an example, for the **Customers** stock variable in Figure 5.5, the inflow (Customer

Acquisition Rate) is calculated with respect to Extent of Requirements Fulfilled, Rework and Features Released that are positive influences. Number of Escaped Bugs positively influences the outflow (Customer Loss Rate), and decreases the inflow. Similarly, the outflow variable (Customer Loss Rate) is positively affected by Number of Escaped Bugs and negatively affected by Extent of Requirements Fulfilled, Rework and Features Released.

5.4 Relationships among Variables

In this section we define the relationships among the variables in the stock flow diagram (Fig 5.5) by defining equations that address their accumulation and depletion in Table 5.1.

Table 5.2 Variables and Equations

Variable Name	**Equation**	**Unit**
Customers (Stock Variable)	Initial Number of Customers + $\int_0^T (Customer\ Acquisition\ Rate - Customer\ Loss\ Rate) dt$	Number of People

Table 5.2 Variables and Equations continued

Customer Acquisition Rate (Inflow)	Extent of Requirements Fulfilled * Features Released * (1-Number of Escaped Bugs)/Number of Bugs before testing) * CONSTANT The CONSTANT has a dimension of People and is determined by the average number of customers who are starting to use the product per unit time.	Number of People per Unit Time
Customer Loss Rate (Outflow)	(1-Extent of Requirements Fulfilled) * (1-Rework/Development Time)*(Number of Escaped Bugs * Significance of Escaped Bugs)*Customers	Number of People per Unit Time
Number of Bugs	Quality of Interface Specification * Source Code Size * CONSTANT The value of the CONSTANT is determined by the average number of bugs normally found in the product by the quality control team per module. The unit is Bugs per Module/Number of Commits)	Bugs per Module

Table 5.2 Variables and Equations continued

Number of Escaped Bugs	Number of Bugs * (1-Test Coverage) Test Coverage is a Constant value between 0 and 1 which has been defined after this table.	Bugs Per Module
Extent of Requirements Fulfilled	(New Feature Requests/Features Released) * 100	Percent/Fraction

Rework	IF THEN ELSE Variable
	IF (Number of Bugs >2 and Customer Loss Rate >10 per unit time) then
	Rework = Available Fixing Resources * Time
	ELSE
	Rework = 0
		Number of Hours

Productivity	Schedule Pressure * Quality of Modularization * Number of Commits	Number of Commits/Unit Time
Quality of Modularization	Ranges from 0 to 1	Dimensionless

Table 5.2 Variables and Equations continued

Quality of Interface Specification	Ranges from 0 to 1	Dimensionless
Features Under Development (Stock Variable)	$\int_{O}^{T} (Features\ Pushed\ into\ Development\ Rate - Development\ Completion\ Rate - Feature\ Deletion\ Rate) dt$	Number of Features
Features Pushed into Development Rate (Inflow)	(1-Schedule Pressure) * New Feature Requests	Number of Features Per Unit Time
Development Completion Rate (Outflow)	Productivity * Number of Features * 1/Number of Commits	Number of Features Per Unit Time
Feature Deletion Rate (Outflow)	Schedule Pressure * Features under Development	Number of Features Per Unit Time

Table 5.2 Variables and Equations continued

New Feature Requests	IF THEN ELSE variable (Depends on customer requests and analysis of Domain Experts)
	IF (Customers > 100) then
	New Feature Requests = 5
	ELSE
	New Feature Requests = 1
		Number of Features

Schedule Pressure	Ranges from 0 to 1	Dimensionless
Features Released	Development Completion Rate * Productivity	Number of Features
Communication Overhead (Stock Variable)	Initial Number of communication Problems + $\int_O^T (Communication\ Problem\ Rate - Resolution\ Rate\ of\ Communication\ Proble$	Number of Problems
Communication Problem Rate (Inflow)	(1-Quality of Interface Specification) * Number of Sites * (1-Cultural Similarity among Sites)	Number of Problems per Unit Time

Table 5.2 Variables and Equations continued

Resolution Rate of Communication Problems (Outflow)	(Quality of Interface Specification) * Cultural Similarity b/w Sites * Used Number of Communication Media	Number of Problems per Unit Time
Number of Distributed Sites	Fixed Variable. Depends on the sites participating in a particular project.	Number of Sites
Number of Communication Media	Fixed Variable. Depends on the communication of the project team. Typically about 3-4 different communication media are used.	Number of Media
Cultural Similarity among Sites	Ranges from 0 to 1	Dimensionless
Available Trained Resources (Stock	Initial Number of Trained Resources + $\int_0^T (Resource\ Training - Resource\ Relocation\ Rate) dt$	Number of People

Variable)		

Table 5.2 Variables and Equations continued

Resource Training Rate (Inflow)	Benched Recruits * CONSTANT Value of the CONSTANT is a fraction and depends on the training capability of the responsible team.	Number of People per Unit Time

Resource Relocation Rate (Outflow)	IF THEN ELSE VARIABLE
	IF (Customer Loss Rate > 10 per unit time) then
	Resource Relocation Rate = 2
	ELSE
	Resource Relocation Rate = 0
		Number of People per Unit Time

Benched Recruits	CONSTANT. Depends on the hiring rate of the organization	Number of People

In table 5.2, we have used a set of new variables to define equations among variables defined the Causal Loop Diagram and the Stock Flow Diagram. They are defined as follows:

- Test Coverage: A qualitative variable representing the number of bugs found and the product code paths covered after unit testing, black box testing, white box testing, integration testing, regression testing etc.

- Source Code Size: The size of the source code base of the whole software product. It is measured in Number of Commits.
- Development Time: The total number of hours put in by human resources per cycle.
- Significance of Escaped Bugs: A qualitative variable that represents the importance of the escaped error in the launched software/product and the extent to which it affects the customer. It ranges from 0 to 1.

CHAPTER 6. RESULTS, ANALYSIS AND CHALLENGES

In this chapter, we will analyze the model that we have proposed, assess the results, alternatives that can be included, the challenges we faced during the course of our study and how they were addressed.

6.1 Results

After the completion of the model we realized that the data we collected from our course project was not enough. Our course project was conducted in an education environment where the participants were full/per time students and we weren't able to collect all the metrics required to be supplied for the environment variables in our model. We analyzed a similar distributed development project [17] and took some of the metrics as a base for our simulation. The initial values for the variables in our model are given in Table 6.1.

Table 6.1 Initial Values of Variables

Variable Name	**Initial Values**	**Unit**
Customers (Stock Variable)	200	Number of People
Customer Acquisition Rate (Inflow)	10	Number of People per Unit Time
Customer Loss Rate (Outflow)	5	Number of People per Unit Time

Table 6.1 Initial Values of Variables continued

Number of Bugs	10	Bugs per Module
CONSTANT (Used for calculating the number of bugs in the Equations Table 5.2)	10	(Bugs per Module)/(Number of Commits)
Number of Escaped Bugs	2	Bugs Per Module
Extent of Requirements Fulfilled	85% or 85/100	Percent/Fraction
Rework	400	Number of Hours
Productivity	75	Number of Commits/Unit Time
Quality of Modularization	0.7	Dimensionless
Quality of Interface Specification	0.7	Dimensionless
Features Under Development (Stock Variable)	6	Number of Features
Features Pushed into Development Rate (Inflow)	3	Number of Features Per Unit Time
Development Completion Rate (Outflow)	2	Number of Features Per Unit Time

Table 6.1 Initial Values of Variables continued

Feature Deletion Rate (Outflow)	1	Number of Features Per Unit Time
New Feature Requests	2	Number of Features
Schedule Pressure	0.8	Dimensionless
Features Released	5	Number of Features
Communication Overhead (Stock Variable)	11	Number of Problems
Communication Problem Rate (Inflow)	2	Number of Problems per Unit Time
Resolution Rate of Communication Problems (Outflow)	2	Number of Problems per Unit Time
Number of Distributed Sites	5	Number of Sites
Number of Communication Media	5	Number of Media
Cultural Similarity among Sites	0.6	Dimensionless

Table 6.1 Initial Values of Variables continued

Available Trained Resources (Stock Variable)	50	Number of People
Resource Training Rate (Inflow)	3	Number of People per Unit Time
Resource Relocation Rate (Outflow)	2	Number of People per Unit Time
Benched Recruits	4	Number of People
Test Coverage	0.7	Dimensionless
Source Code Size	300	Number of Commits
Significance of Escaped Bug	0.5	Dimensionless
Development Time	8000	Number of Hours

The simulation results for the stock variables Customers, Features under Development, Communication Overhead and Available Trained Resources are shown in Figures 6.1, 6.2, 6.3, 6.4 respectively. In all the figures, the X axis represents a time period of 24 months over which the simulation has been run. The Y axis represents the stock variable for the specific graph as defined in Table 5.2.

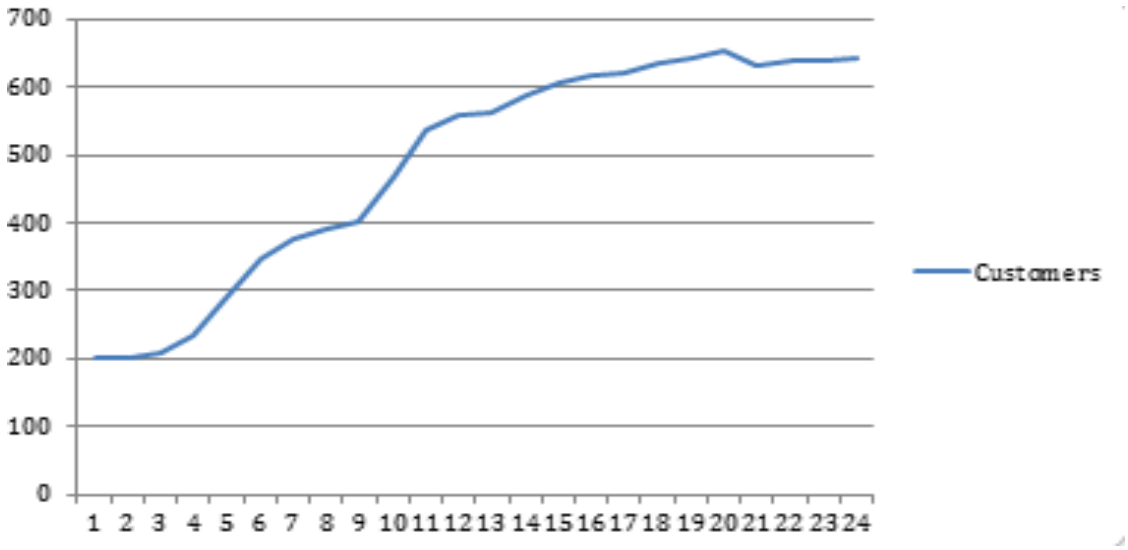

Figure 6.1 Predicted number of Customers

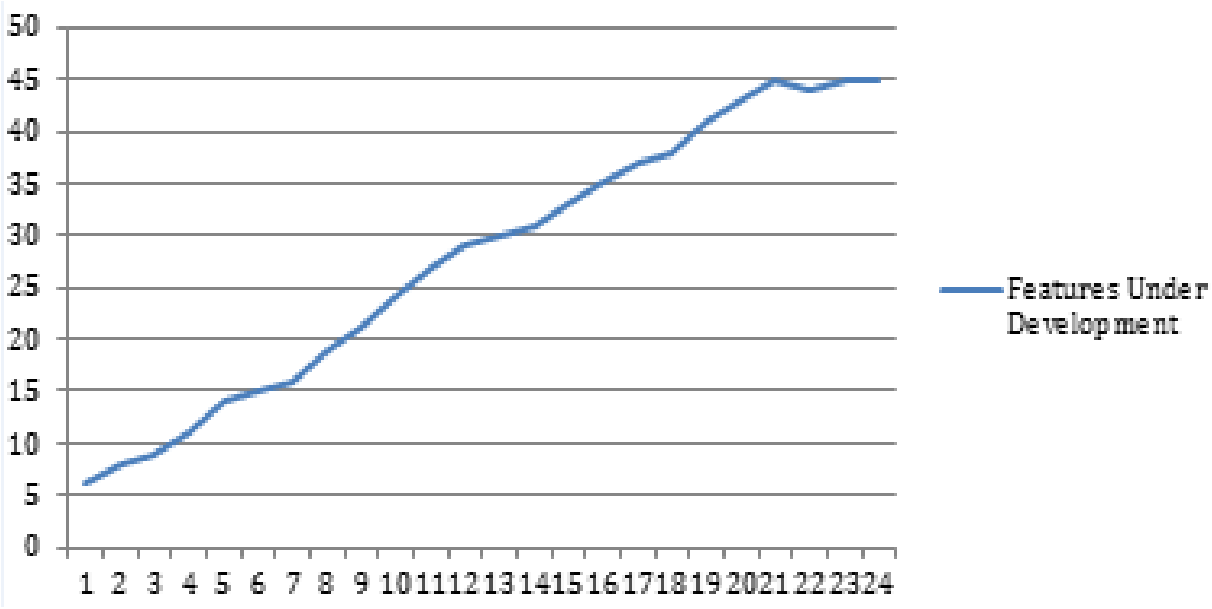

Figure 6.2 Predicted number of Features Under Development

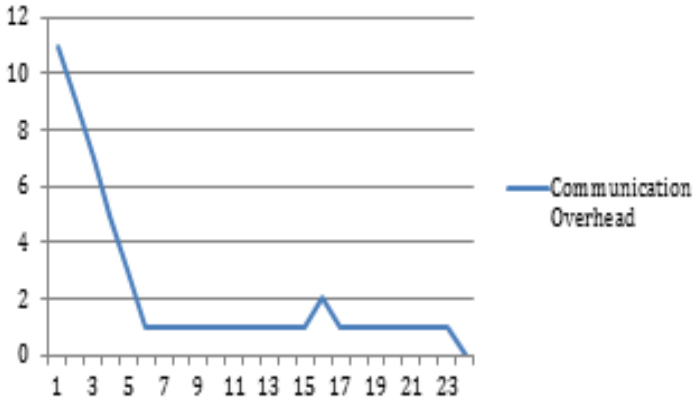

Figure 6.3 Predicted number of Communication Overheads

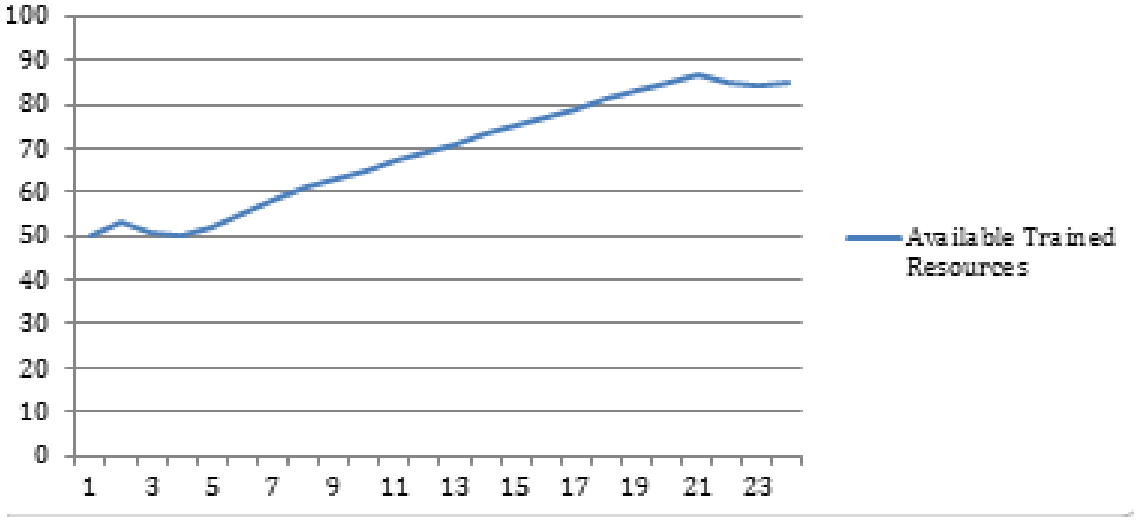

Figure 6.4 Predicted number of Available Trained Resources

6.2 Analysis & Alternative Variables

The model presented in the previous chapter was created with reference to a globally distributed software development environment that we observed over two iterations of the course project (Fall 2013, Fall 2014). We analyzed the deliverables, the participants, different problems and solutions in order to identify the system variables or components that were best for the domain we

described in Chapter 3. The results indicate the values of the stock variables over a period of 24 months. We spoke to multiple researchers [18. 19. 20] in the field of software development and took their feedback about the variables we had identified, the relationships that we had defined among variables and improved our model further. Needless to say, an industry standard software development environment has a much bigger scale. The number of resources involved, deliverables, product cycles and customer base is much larger. Depending on the project, a number of other stock variables can be included in similar projects such as:

- Customer Satisfaction
- Testing & Quality Control
- Maintenance & Support
- Capability of Scaling Performances of the Software
- Efficiency of Each Development Site
- Revenue of the Organization

Furthermore, including these stock variables into the model will entail more auxiliary variables and dependencies. Various software companies can identify their own development processes with the help of domain experts and create models for themselves by supplying data from their past projects. After a model has been created and simulated, it can detect effects of changes of system variables or unforeseen modifications. Finally, various measures can be taken to adapt better, earlier and nullify their unwanted impacts to stock variables and other stakeholders.

6.3 Challenges

During the course of the observation and data collection for this project, several problems arose that affected the accuracy of the model. One of the main problems that we faced was in the process of data collection. In Fall 2013, the manager of each team was responsible to collect data from his team members each week. At times, he forgot or some of his teammates weren't punctual. This process was improved in Fall 2014 where a real time service such as Google Docs was used which provided instant feedback. Using this, people who were submitting data could be tracked and can be contacted directly in the event of a delay. Team members were contacted directly in case of aberrations to validate what was submitted.

Another uncertainty that we faced was related to the significance of the course project under observation. Although it was distributed in nature, the participants in Iowa State University teams were full time students and some in the other universities were part time students who have other jobs as well. We tried to ensure the precision of the system by providing a deliverable schedule that mirrored a real life software development lifecycle, followed standard development procedures, faced similar problems and took measures to mitigate them.

CHAPTER 7. CONCLUSION & FUTURE WORK

In our research, we identified the main system variables, stakeholders and issues during two iterations of the Distributed Software Development course that took place across four countries. We analyzed the behavior of participants, schedule of deliverables, various communication issues and responsible causes. This helped us identify the relationships between each component and provide numerical equations to represent them. We presented the process of building a system dynamics model with the identified variables, presented the results and also recommended different alternatives to suit various organizations, work processes and their interests.

One of the foremost problems that we faced during the course of our research was the lack of data from industry. Although we got the model reviewed and made changes according to recommendations from professors and lead software developers in the field, we still didn't get to draw similarities with our model and live data from real life software projects. Our model can be improved further in the future by involving other related stock, auxiliary variables and collecting data from real software projects involving full time human resources. This would help us create more relevant models and simulate the model against the backdrop of data supplied from the industry. Finally using simulated models, we can detect changes in the system and help managers to take preventive

measures to lessen the effects of changes and to make informed decisions to improve development methodology, product quality and values of stock variables for the system.

REFERENCES

[1] J.D Herbsleb and D. Moitra (2001), Global Software Development, IEEE Software, March/April, USA, p. 16-20.

[2] A Gupta and S Seshasai (2007), The Critical Role of Information Resource Management in Enabling the 24- Hour Knowledge Factory (September 2, 2007). Available at SSRN: http://ssrn.com/abstract=1011417 .

[3] B. Sengupta, S. Chandra & V. Sinha, A Research for Agenda for Distributed Software Development, Proceeings of 28th International Conference on Software Engineering, Shanghai, China, 2006.

[4] J. Herbsleb and A. Mockus, An empirical study of speed and communication in globally distributed software development. IEEE Transactions on Software Engineering, 29(6):481–94, 2003.

[5] N Ramasubbu and R.K Balan, Globally Distributed Software Development Project Performance: An Empirical Analysis. In Proceedings of the 6th Joint Meeting of European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE '07), 125-134. New York: ACM. Doi: 10.1145/1287624.1287646.

[6] J.D Sterman, System Dynamics Modeling for Project Management. MIT Press, Cambridge, MA available at: www.web.mit.edu/jsterman/www/SDG/project.html.

[7] J.A Espinosa, S Slaughter, J Herbsleb & R Kraut, Team Knowledge and Coordination in Geographically Distributed Software Development. International Conference on Information Systems (ICIS), 2001, New Orleans.

[8] H Rahmanad and D Weiss, Dynamics of Concurrent Software Development. System Dynamics Review, 25(3) (2009), pp. 224-249.

[9] S Murthy, R Gujrati and S Iyer, Using System Dynamics to Model and Analyze a Distance Education Program. In Proceedings of the International Conference on Information Technologies Development, 2010, London, United Kingom.

[10] http://vensim.com/vensim-personal-learning-edition/

[11] V. Basili, G. Caldiera and H.D. Rombach, Goal Question Metric Aproach. Encyclopedia of Software Engineering, pp. 469-476, John Wiley & Sons, Inc., 1994.

[12] V. Basili and D. Weiss, Evaluation of a Software Requirements Document by Analysis of Change Data, Proc. 5th International Conference on Software Engineering, March 1981.

[13] V. Basili and D. Weiss, A methodology for Collecting Valid Software Engineering Data, IEEE Trans. on Software Engineering, November, 1984.

[14] B.J. Angerhofer an M.C. Angelides, System Dynamics Modeling in Supply Chain Management: Research Review. In Proceedings of the 32nd Conference on Winter Simulation, 2000, pp. 342-351.

[15] G.P. Richardson and P. Otto, Applications of System Dynamics in Marketing: Editorial. Journal of Business Research, 2008, vol. 61, issue 11, pp. 1099-1101.

[16] N Ghaffarzadegan, J. Lyneis and G.P. Richardson, How Small System Dynamics Models Can Help the Public Policy Process, 2010, System Dynamics Review, vol 27, issue 1, pp. 22-44.

[17] R.L. Hackbarth, A. Mockus, J.D. Palframan and D. Weiss, Assessing the State of Software in Large Enterprise. Empirical Software Engineering, June 2010, Volume 15, Issue 3, pp 219-249.

[18] D.M. Weiss; Personal Communication, December 2014-April 2015.

[19] Robert Ward; Email Communication, 26th December, 2014.

[20] James Houghton; Email Communication, 10th February, 2015.

[21] D Parnas, P Clements, D Weiss: The Modular Structure of Complex Systems. Proceedings of 7th International Conference on Software Engineering, March 1984.